# Efficient parallel message computation for MAP inference

Stavros Alchatzidis, Aristeidis Sotiras, Nikos Paragios

# Efficient Parallel Message Computation for MAP Inference

Stavros Alchatzidis Aristeidis Sotiras Nikos Paragios *
Center for Visual Computing, Ecole Centrale de Paris, Châtenay-Malabry, France
Equipe GALEN, INRIA Saclay, Île-de-France, Orsay, France

## Abstract

*First order Markov Random Fields (MRFs) have become a predominant tool in Computer Vision over the past decade. Such a success was mostly due to the development of efficient optimization algorithms both in terms of speed as well as in terms of optimality properties. Message passing algorithms are among the most popular methods due to their good performance for a wide range of pairwise potential functions (PPFs) . Their main bottleneck is computational complexity. In this paper, we revisit message computation as a distance transformation using a more formal setting than [8] to generalize it to arbitrary PPFs. The method is based on [20] yielding accurate results for a specific class of PPFs and in most other cases a close approximation. The proposed algorithm is parallel and thus enables us to fully take advantage of the computational power of parallel processing architectures. The proposed scheme coupled with an efficient belief propagation algorithm [8] and implemented on a massively parallel coprocessor provides results as accurate as state of the art inference methods, though is in general one order of magnitude faster in terms of speed.*

## 1. Introduction

Markov Random Fields were initially introduced in computer vision to address image restoration [9] and have been considered to address more complex problems like segmentation [22], stereo-reconstruction [23], image registration [10] etc. Key part for the success of such models has played the advance made on the conception of efficient optimization algorithms. Initially, the main shortcoming of the inference algorithms [3, 6] was either their slow convergence or the lack of optimality guarantees on the obtained solutions. These shortcomings were alleviated by the introduction of techniques like graph-cuts [11], belief propagation [19] and more recently linear programing methods [15] that boosted the interest towards MRF models.

Efficient inference on such models can be performed either by graph-based methods or message passing ones. The first ones are based on the max-flow min-cut principle and exhibit high computational efficiency, especially when applied to regular graphs. Graph-cut methods [11] belong to this class as well as their multi-label [4, 5] and dynamic graph-cut extensions. Their main drawback is that they are limited by the type of energy to be minimized [13]. The second class of methods is based on propagation of beliefs in the graph by local exchange of messages. Max-Product Loopy Belief Propagation [17], its efficient variants [8, 23], tree-reweighted message passing [12] and more recently dual decomposition [14] are representative methods in this direction. These methods are able to cope with arbitrary energies. Moreover, they tend to provide higher quality solutions and better optimality bounds [12, 14] while at the same time being able to handle high-order interactions [16].

Despite the success of message passing algorithms, they rest computationally demanding, a fact that compromises their use in large graphs with large label-sets. To counter this, an efficient message-passing computation as a distance transformation was proposed in [8]. The distance transform is performed in a sequential way. Belief propagation has also been investigated in Graphical Processing Units (GPU) [26] in an effort to accelerate the inference throught the computational power of the parallel architecture.

In this paper, we introduce a novel method to estimate the message costs based on the Jump Flooding concept [20]. The main strengths of this method are: i) its capacity for parallel implementation, ii) its generality w.r.t. pairwise energy types, iii) its feasible scaling $O(nlogn)$ with the number of candidate labels for a node. This method has been incorporated to a state-of-the-art optimization algorithm and has been implemented in GPU leading to decreased running times while being able to capture good minima.

The remainder of this paper is organized as follows: in section 2, we briefly review the state-of-the-art on belief propagation methods and in particular the one introduced

*The work was partially supported by the European Community's Seventh Framework Programme, ERC grant 259112 (DIOCLES) and the sterEOS+ grant of the Medicen Ile-de-France Competitive Cluster. S. Alchatzidis was partially supported by the Greek State Scholarships Foundation. Contact author: stavros.alchatzidis@ecp.fr

in [8]. In section 3, we formulate message computation as a distance transformation. Following, in section 4, the novel parallel message computation scheme is presented. Experimental results on the Middlebury MRF benchmark [1] are presented in section 5, while section 6 concludes the paper.

## 2. Belief propagation methods

The discrete MRF problem is an attempt to assign to each node p of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a label $l_p$ coming from a label-set $\mathcal{L}$. $\mathcal{V}$ and $\mathcal{E}$ denote the set of the vertices and the edges of the graph respectively. The set of vertices models the variables to be estimated, while the one of edges the interactions between them. The labels correspond to the possible values that the latent variables can take. The labeling problem can be cast as the following minimization one:

$$\min \sum_{p \in \mathcal{V}} d_p(l_p) + \sum_{p,q \in \mathcal{E}} d_{pq}(l_p, l_q), \tag{1}$$

where $d_p$ represents the unary potentials and $d_{pq}$ the pairwise ones.

Belief propagation methods attempt to optimize this problem by exchanging messages between nodes. Each message is a vector with size equal to $K = |\mathcal{L}|$ (by $|\cdot|$, the cardinality of the set is denoted). We define as $m^t_{p \to q}$ the message that is transmitted from node $p$ to node $q$ at iteration $t$. At each iteration messages are computed as follows:

$$m^t_{p \to q}(l_q) = \min_{l_p}(d_{pq}(l_p, l_q) + \underbrace{d_p(l_q) + \sum_{n \in \mathcal{N}(p) \setminus q} m^{t-1}_{n \to p}(l_p)}_{I}), \tag{2}$$

where $\mathcal{N}(p)$ is the set of nodes with which node $q$ is connected with an edge (also called neighborhood nodes). Hereafter, the message that results from the previous operation will be also denoted as $m_{res}$. Note that in the previous equation as well as in the rest of our analysis and implementation, negative **log** probabilities are considered turning the initial max-product problem into its equivalent min-sum one. A normalization step is usually applied afterwards contributing to the robustness of the algorithm:

$$m_{res}(l_p) = m_{res}(l_p) - \min_{l_q} m_{res}(l_q) \tag{3}$$

At each iteration, a belief vector $b$ can be computed for every node,

$$b_q(l_q) = d_q(l_q) + \sum_{n \in \mathcal{N}(p)} m^t_{n \to q}(l_q) \tag{4}$$

The labeling that corresponds to Maximum A-Posteriori Probability estimation for a node $p$ is thus given by the labels that minimize the belief, or:

$$\min_{l_p} b_p(l_p) \tag{5}$$

### 2.1. Belief propagation networks and prior art

Pearl in [19] introduced the method for inference on Bayesian Networks. The proposed sum-product algorithm provided exact marginal probabilities when applied to acyclic graphs by exchanging two messages per edge. Murphy et al. [17] proposed the use of the BP algorithm even in graphs containing cycles. They showed that although it did not converge to the global MAP solution, it produced unexpectedly good results for different kinds of problems. Tappen et al. [23] having noticed the inefficiency in the propagation of information proposed accelerating it by propagating first over rows and then over columns. Kolmogorov [12] introduced the TRW-S algorithm extending the approach taken by Wainwright et al. [25], creating a sequential algorithm which guaranteed the convergence towards a good solution.

Our implementation is based upon the variant proposed by Felzenszwalb et al. [8] (referred hereafter as $BP - P$). In their paper, three ways were proposed to speed up the Loopy-BP algorithm:

i) a multiscale approach, combining unary costs to derive a higher level graph and using the resulting messages to initialize the lower level graph messages;

ii) a checkerboard message computation scheme, computing in turns white and black tiles thus increasing propagation speed and halving memory requirements. Here, "checkerboard" stands as a metaphor for a graph in grid connectivity;

iii) a distance transform approach to message computation, resulting in algorithms with lower computation complexities for special classes of pairwise potentials.

The last contribution has proven to be the most popular, being incorporated in the implementations of many algorithms, allowing them to achieve great speed improvements. The main shortcomings of this method are the non-generalization to other pairwise potentials and the sequential nature of the message computation. The latter, renders problematic the design of implementations able to take advantage of the emerging multiprocessor computer architectures. By reinterpreting message computation as a distance transformation in a more general theoretical framework, we address them both by introducing a new parallel algorithm. In addition, we provide a GPU implementation to exhibit its advantages in efficiency.

## 3. Message computation

### 3.1. General message computation

The general message computation, as defined in equation (2), can be computed in 3 steps:

- addition of the message vectors and the unary potential vector of the node. The result is an intermediate vector $I$

- a double loop, calculating for each of the positions (labels) of the resulting message $m_{res}$ the minimum value of the intermediate vector when added to the pairwise potential

- normalization, as defined in equation (3).

The second step consumes most of the execution time and its complexity $O(n^2)$ forbids the application of Belief Propagation methods to large label-sets. We will refer from now on to step 2 as message computation (MC) as it constitutes the most essential part of it.

Felzenszwalb et al. in [8] introduced message computation as a distance transformation. For completeness reasons, their work is going to be presented briefly while more emphasis will be put on the connection between message computation and distance transforms. A connection that will enable us to discuss the properties of the proposed algorithm.

### 3.2. Efficient message computation in BP-P

In BP-P two algorithms are proposed to compute messages efficiently for two types of pairwise potentials: the $L_1$ norm, $d_{pq}(l_p, l_q) = |l_p - l_q|$ and the quadratic distance $d_{pq}(l_p, l_q) = (l_p - l_q)^2$. For linear pairwise costs, the algorithm scans twice $I$ updating its values sequentially and requires no further buffer (see Fig. 1). For the quadratic ones, the algorithm computes the lower envelope of parabolas, $(x - l_p)^2$ requiring 3 extra buffers to hold information about intersections, and positions of the parabolas participating in the lower envelope. Both algorithms exhibit an $O(n)$ complexity and are not susceptible to any obvious parallelization scheme.

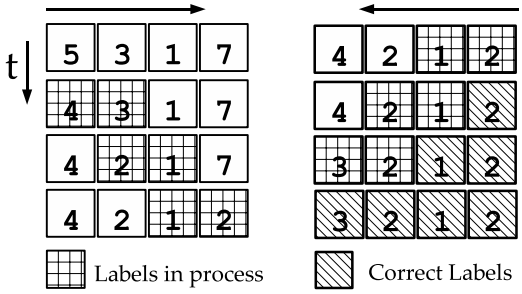


Figure 1. The algorithm utilized by BP-P for $L_1$ norm pairwise costs. Left: forward pass. Right: backward pass

### 3.3. Message computation as a distance tranformation

Message computation can be regarded as an *additively weighted distance transformation with an unknown number of seeds*. We will use the distance transformation terminology to further analyze the problem at hand. In the specific context, distance is defined by:

$$d_{AWDT}(k, l) = I(k) + d_{pq}(k, l), \tag{6}$$

where again $I$ stands for the intermediate vector and $d_{pq}$ for the pairwise potential function.

An *area of influence* (AOI) of a label $i$ is a set that consists of all the labels that are closer (in terms of the previous distance) to label $i$ than any other label. Label $i$ is called *seed*.

$$AOI_i = \{l : \arg\min_k d_{AWDT}(k, l) = i\}. \tag{7}$$

The value of the message $m_{res}$ for a label $l$ depends on the seed to whose AOI the label belongs, or:

$$m_{res}(l) = I(i) + d_{pq}(i, l),\ l \in AOI_i. \tag{8}$$

Let $\mathcal{L} = (1, \ldots, n)$ be the set of all labels. For the set of all the AOIs the following should stand:

$$\cup\, AOI_i = \mathcal{L}\ and\ AOI_i \cap AOI_j = \varnothing, \tag{9}$$

or a label can belong to only one AOI.

Thus, MC can be seen as the process which, given an intermediate vector $I$ and a label-set $\mathcal{L}$, results in a set of seeds $\Sigma$ (because initially any label can be a seed) and their respective AOIs:

$$I \xrightarrow{MC} \{AOI_i, i\},\ i \in \Sigma \tag{10}$$

**Lemma 1:** If $d$ is a metric than $i \in AOI_i$.

**Proof:** If $i$ does not belong to its own AOI it will belong to another one's. Let this label be $l$, then

$$I(l) + d(i, l) < I(i), \tag{11}$$

as $d(i, i) = 0$, given $d$ is a metric. If $i$ is a seed then there exists some label $k$ that belongs to its AOI. So it should stand that:

$$I(i) + d(i, k) < I(l) + d(l, k). \tag{12}$$

By replacing the former equation to the latter, we get:

$$d(i, l) + d(i, k) < d(l, k), \tag{13}$$

which contradicts with the definition of a metric. Thus, we can deduce that $i \in AOI_i$ since no such label $l$ can exist.

## 4. Message computation using the Jump Flooding algorithm

The main idea proposed in this paper is to use the Jump Flooding algorithm to perform MC in parallel. The Jump Flooding algorithm was introduced by Danielsson in [7] and after many years reintroduced by Rong et al. [20] as a parallel framework for use on GPUs and especially to calculate Euclidean distance transforms. It is parallel and completely symmetrical. In our case, symmetry solves the difficult issue of the unknown number of seeds as the algorithm treats by design every label as a possible seed.

### 4.1. The JF algorithm

To ease the presentation, the problem is going to be formulated for the $1D$ case, for which our experiments have been held. The algorithm operates on the intermediate vector $I$ (we suppose $K$ a power of 2 without loss of generality) in an iterative manner and terminates after $log_2(K)$ iterations. The algorithm propagates information stored in an auxiliary vector $S$ which holds closest seed correspondences for every label. The vector is initialized at $S_0[k] = k$ (subscript stands for iteration) or initially each label belongs to its own seed. At each iteration $S$ is updated as:

$$S_{i+1}[k] = \underset{n=S_i[k+d],S_i[k-d],S_i[k]}{\arg\min} I(k) + d_{pq}(k,n), \quad (14)$$

where

$$d = 2^{log_2(K)-i}, \quad (15)$$

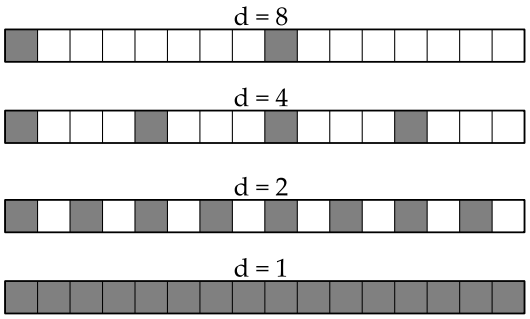


Figure 2. Information propagation in the JF algorithm. Label 0 propagates its information to every other label in 4 steps.

Elaborating, node $k$ propagates $S_i[k]$ to nodes that are situated at $d$ positions away in each direction (e.g at nodes situated at $k + d$ , $k - d$ if such exist). In a symmetrical view of the algorithm, every node $k$ receives information from nodes at distances $d$, compares the information with its own and deduces the $S_{i+1}(k)$.

### 4.2. Errors in the JF algorithm

The JF algorithm is known to be approximate. Inaccuracy is produced by the non-propagation of seed information to all labels that should belong to its AOI. This is mainly related to the geometry of the AOIs of the exact solution. Thus, two questions arise at this point: (1) for which geometries of AOIs can we find accurate (or closely approximate) solutions using our algorithm, and (2) to which geometries a potential pairwise function corresponds.

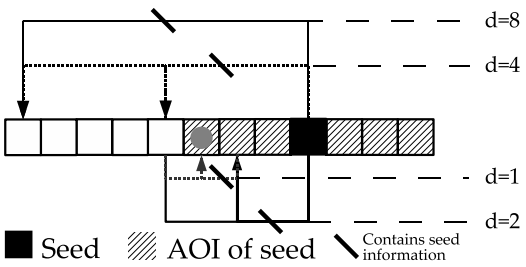


Figure 3. Visualization of seed information propagation in a connected AOI containing its seed. The dotted label will get the seed's information from a propagation path within the AOI.

We claim that our algorithm can find the right solution for a **pairwise distance that produces connected AOIs which contain the seed label** (as we have already seen in Lemma 1, $s \in AOI_s$ for metric pairwise distances). We will prove that every label belonging to such an AOI will be labeled correctly. Let $s$ be the corresponding seed for such an AOI. In general, the following stands:

$$\forall l \in AOI_s : [s,l] \subseteqq AOI_s. \quad (16)$$

$[\cdot,\cdot]$ denotes a closed set. As proven in [20], any label $l$ within the AOI can receive $s$'s information at a certain iteration $i = log_2(K) -$ *position of the leftmost* 1 *of* $|s-l|_2$, from both of its $d$-neighbours (if both of them exist). This information has followed two separate paths. One path passes from labels belonging strictly to $[s,l]$ while the other one passes by at least one label outside of $[s,l]$. For $l$ not to receive the information, both of the paths should have rejected $s$'s information. **That is not possible as one of these paths passes exclusively from labels within** $AOI_s$ which will propagate s. This means that in equation (14) $S_{i+1}[k]$ will be equal to $s$ throughout this path (see Fig. 3).

Ash and Bolker [2] give much insight on the geometry of AOIs, studying their relation with the corresponding weighted distance function in the 2D case. They prove that a 2D additively weighted euclidean distance function produces AOIs separated by hyperbolas, a corresponding quadratic produces convex polygonal AOIs while a logarithmic produces circular ones. The formulation and the intuition developed in this paper can be extended to the 3D case or specialized to the 1D case to extract approximation properties of our algorithm for a given pairwise function. Pairwise function apart, we believe that the connectivity of

the AOI and the inclusion of its seed within it are necessary conditions (and in the 1D case sufficient also) for the algorithm to produce accurate results and any functions not respecting these will produce approximate results.

### 4.3. Approximation effects in the context of MAP inference

In summary, the algorithm makes a distinction between *dominated seeds*, for which $i \notin AOI_i$, and *dominating seeds*, for which $i \in AOI_i$. For the AOI of the second ones, it will find an exact solution while for the AOI of the first ones probably not as correct seed information may not get propagated to the labels of the AOI.

Inference-wise, the effects of this approximation in the general case should not be of much significance. Using an exact scheme, the labels of dominated AOIs would be heavily penalized. In the approximate solution, they will be penalized even more. Thus, in both cases, their contribution to the determination of the labeling of a smooth area is of minimal importance. In such areas, the optimal labeling of neighboring pixels varies little. As a result, the labels that should be considered to refine the labeling in subsequent BP iterations come from AOIs with exact values that overlap. Thus, the optimal label can be decided with precision.

Approximation errors occur at the meeting lines of smooth areas (also called edges) where the use of a truncated distance or one that reaches fairly quickly a plateau (like the log function), results in an abrupt change in the labeling of neighboring nodes. In such areas, the optimal labeling of neighboring nodes comes from non-overlapping AOIs. As a result, the computation of the intermediate vector is based on the combination of both exact and over-penalized information. This may alter the order of importance of the seeds and thus lead to an inexact labeling.

Extensive validation has led us to believe that this error occurs rarely and in any case it has strictly local effects. In the truncated quadratic case (a non-metric distance) and using the penguin image for denoising we observe that the approximation effects don't influence much the quality of the optimization as depicted in Fig. 4. There, the evolution of the energy over time is similar for both the exact and approximate approaches. In Fig. 5, we can see the evolution of the average absolute error per label. Given the optimization parameters (weighting factor $\lambda = 25$ and truncation equal to 200 resulting in a maximum label error equal to 5000) the error is minimal. The above is supported by the fact that when the mean absolute error gets values around 1, the corresponding energy coincides with the exact one.

## 5. Experimental results

The main aim of the validation is to show the merits of the proposed implementation in two domains: speed and optimization quality. More specifically, we compare with two state-of-the-art algorithms (alpha-expansion [5], TRW-S [12]) to show that our algorithm provides a very competitive solution within a very small time interval. Moreover, by comparing to the BP-P variants running on the same hardware, we demonstrate the utility of the addition of a parallelization scheme for MC to an algorithm that can already undergo parallelization in higher levels.

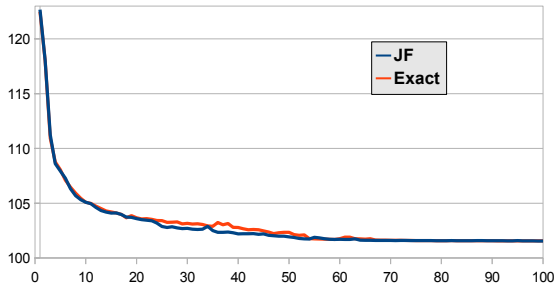


Figure 4. Energy - Iteration comparison of the exact and approximate methods for quadratic pairwise energy optimization.

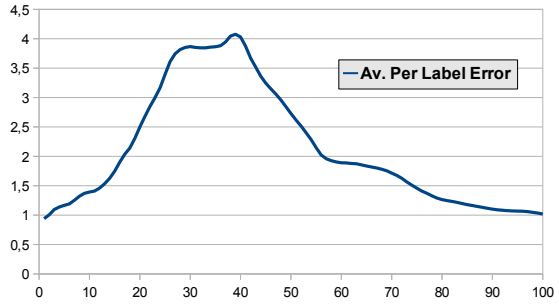


Figure 5. Average absolute error per label - Iteration for quadratic energy optimization. Average error per label is computed by comparing the corresponding messages of the two algorithms (JF and Exact) on every iteration.

Our validation is based on the Middlebury MRF benchmark [1]. Three different test cases are used:

- Denoising on the Penguin image using a truncated quadratic pairwise energy ($\lambda = 25$, $truncation = 200$, $|\mathcal{L}| = 256$);
- Denoising on the Penguin image using a t-student pairwise energy ($\alpha = 700$, $\sigma = 7$, $|\mathcal{L}| = 256$);
- Stereo on the Tsukuba image pair using a truncated linear pairwise energy ($\lambda = 20$, $truncation = 2$, $|\mathcal{L}| = 256$).

### 5.1. Comparison with state-of-the-art methods

We created a GPU implementation of our algorithm and tested it against TRW-S (for quality reference) and alpha-expansion (for speed reference). The lower bound of the

| Algorithm | Abs.Time | ×Faster |
|---|---|---|
| Efficient | 22.2s | ×71.2 |
| Naive | 1582s | ×1 |
| JF | 52s | ×30.42 |

Table 1. Running times of CPU BP implementations after 100 iterations using a quadratic pairwise energy. See sec. 5.1 for details.

energy provided by TRW-S is used as the baseline with respect to which the rest of the results are given as in [1]. The GPU implementations run on a 256 core coprocessor with an 89.6 GB/s device memory bandwidth and the CPU versions on a quad-core Intel Xeon W3530 working at 2.8Ghz. The GPU has been exploited using Nvidia's CUDA API. Every algorithm run for 100 iterations except from TRW-S that run for 200 in order to get an accurate approximation of the lower bound of the energy.

In general, our implementation delivers high-quality solutions in very small time intervals. As witnessed by Fig. 6 our implementation dominates alpha-expansion both qualitatively as well as in terms of speed while it can provide a solution with energy around 101.5 percent of the optimal one in the 1/15 of the time required by TRW-S to provide an equivalent one (not shown in the diagram). In Fig. 8 we see that although both alpha-expansion and TRW-S give better quality solutions, our algorithm can provide its 101 percent energy solution in about half the time.

**T-student potentials**. One of the main strengths of the proposed framework is its applicability to general pairwise functions. To demonstrate this, apart from the 'traditional' linear and quadratic pairwise terms we included tests using the t-student potential introduced in [24] and used in the celebrated [21]. Indeed, as seen in Fig. 7 our algorithm outperforms by far other algorithms while retaining great quality properties (reaching almost the global minimum).

### 5.2. Comparison with BP-P variants

To further point out the interest of the proposed method, we compare it to BP-P variants running on the same hardware. First, we consider single processor architectures even though one of the main advantages of our method is its suitability for parallel implementation. All the BP-P CPU implementations have been parallelized using OpenMP.

We claim that lower complexity renders it a fast alternative to the naive $O(n^2)$ MC algorithm. As depicted in Table 1, our method achieves a considerable speed-up comparing to the naive scheme though being naturally slower than the efficient $O(n)$ implementation. **Nonetheless, we should point out the ×30 speedup achieved for distances that have no efficient scheme of computation as is the t-student.**

To show the utility of our parallelization scheme, we also implemented two GPU versions of BP-P. One using efficient MC algorithms (referred hereafter as $(GPU\_BP-P)$), and another using the naive $O(n^2)$ algorithm $((GPU\_BP-naive))$. In the next paragraphs, knowledge of the notions of shared (on-chip) and global(off-chip) memory are supposed as well as a level of acquaintance with the SIMT (Single Instruction Multiple Threads) architecture. For further reading please refer to [18].

| Algorithm | Abs.Time | × Faster |
|---|---|---|
| Naive | 1584s | ×1 |
| JF | 54s | ×29.33 |

Table 2. Running times of CPU BP implementations after 100 iterations using a t-student pairwise energy.

| Algorithm | Abs.Time | ×Faster |
|---|---|---|
| Efficient | 137s | ×0.204 |
| Naive | 28s | ×1 |
| JF | 5.21s | ×5.3 |

Table 3. Running times of GPU BP implementations after 100 iterations using a quadratic pairwise energy. See sec. 5.1 for details.

| Algorithm | Abs.Time | × Faster |
|---|---|---|
| Naive | 207s | ×1 |
| JF | 14s | ×14.78 |

Table 4. Running times of GPU BP implementations after 100 iterations using a t-student pairwise energy.

The implementation of the two efficient algorithms proposed in [8] displays perfectly the fitness of our method to an SIMD architecture. A straightforward implementation scheme of MC using shared memory (having excluded in advance one solely based on global-memory), would be to assign a thread to every message and use the efficient variants where applicable. In practice this approach fails as the message length increases.

Using sequential algorithms on a parallel machine is principally wrong. The main bottleneck of massively parallel processing architectures is memory-throughput. This can be seen as how can we keep all of our processors busy having in mind that we have to access data from the slow, off-chip global memory. Remember that CPUs use a large memory hierarchy in order to hide this latency. GPUs trade large memory hierarchies with increased core numbers. So, if a CPU computation required $x$ bytes of data loaded in the cache hierarchy we can not just apply it blindly to a GPU as it would now require $N \times x$ bytes where $N$ the number of cores per multiprocessor.

In the CUDA case a straightforward implementation translates to a lack of available shared memory to each multiprosessor. This results in multiprocessors being more underutilised the more lengthy messages become. This effect

| Algorithm | Abs.Time | ×Faster |
|---|---|---|
| Efficient | 18.94s | ×7.65 |
| Naive | 145s | ×1 |
| JF | 19s | x7.63 |

Table 5. Running times of GPU BP implementations after 100 iterations using a linear pairwise energy. See sec. 5.1 for details.

is greater in the quadratic case where 4 buffers are required. This makes the quadratic efficient implementation slower even than the naive $O(n^2)$ one(see Table 3). On the other hand, the linear case requires only one buffer and is, thus, comparable to our implementation in terms of speed (see Table 5). We should note that even though we kept the message length at 256 for the sake of a clear comparison, for high enough message lengths even the linear efficient algorithm should be slower than our method.

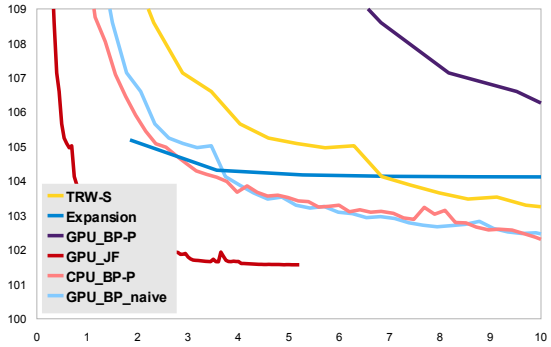


Figure 6. Time-Energy comparison using a quadratic pairwise energy.

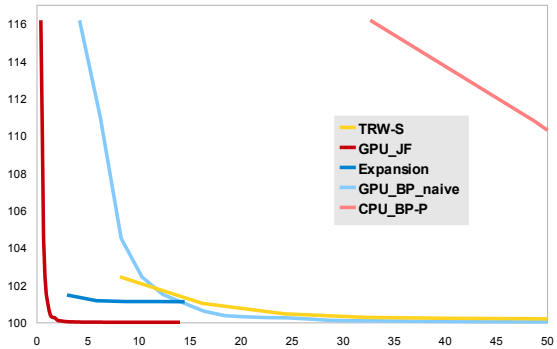


Figure 7. Time-Energy comparison using a t-student pairwise energy. Notice the great efficiency gap between our method and the rest as for this distance there is no efficient computation scheme.

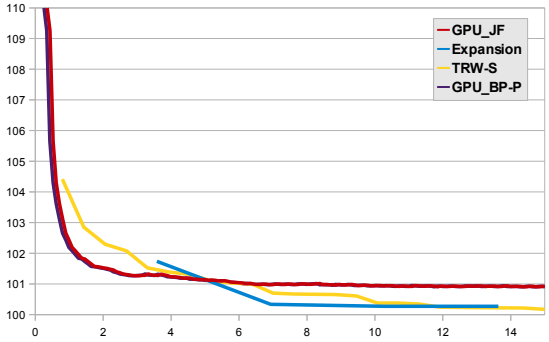


Figure 8. Time-Energy comparison using a linear pairwise energy. Notice that the efficient scheme in the GPU runs slightly faster than our method.

## 6. Discussion

In this paper we have presented a generic message computation scheme for MAP inference on MRFs which resolves two major issues of current efficient implementations: (1) the non-parallelization (2) the non-generalization to pairwise potentials other than the $L_1$-norm and the quadratic. We provide a class of pairwise functions where our results are accurate and an understanding of why results for other classes, though approximate, should be close to the accurate ones. We also provide a GPU implementation of the algorithm in [8] to illustrate the potential of our method both in terms of speed and quality of optimization.

Our work allows for many algorithms based on message passing to take full advantage of recent advances in parallel architectures and corresponding hardware availability. We consider our work valuable in the context of high-order MRF modeling where the problem of large label-sets dominates the inference complexity and the usage of potentials for which there are no efficient calculation methods [16] has led to approximation schemes [21].

## References


[1] S. R. . al. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE T-PAMI*, 30(6):1068 –1080, june 2008. 2, 5, 6

[2] P. F. Ash and E. D. Bolker. Generalized dirichlet tessellations. *Geometriae Dedicata*, 20:209–243, 1986. 4

[3] J. Besag. On the statistical analysis of dirty pictures. *JRSS*, B-48:259–302, 1986. 1

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE T-PAMI*, 26(9):1124 –1137, sept. 2004. 1

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE T-PAMI*, 23(11):1222 –1239, Nov 2001. 1, 5

[6] P. B. Chou and C. M. Brown. The theory and practice of bayesian image labeling. *IJCV*, 4:185–210, 1990. 1

Figure 9. Comparison of t-student and quadratic potential function modeling for inpainting. First row: t-student potentials. Second row: quadratic potentials. First column: CPU BP (top: efficient, bottom: naive). Second column: GPU BP - naive MC. Third column: GPU JF - proposed method. Fourth column: Expansion. Fifth column: TRW-S. Sixth column: Top : Original Image. Bottom: GPU BP-P - efficient message passing (only for the quadratic case).


[7] P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980. 4

[8] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. *IJCV*, 2006. 1, 2, 3, 6, 7

[9] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE T-PAMI*, 6(6):721 –741, nov. 1984. 1

[10] B. Glocker, A. Sotiras, N. Komodakis, and N. Paragios. Deformable medical image registration: setting the state of the art with discrete methods. *Annual review of biomedical engineering*, 13:219–44, aug 2011. 1

[11] D. M. Greig, B. T. Porteous, and A. J. Seheult. Exact minimum a posteriori estimation for binary images. *Journal of the Royal Society. Series B (Methodological)*, 51(2):271–279, 1989. 1

[12] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE T-PAMI*, 28(10):1568–1583, 2006. 1, 2, 5

[13] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts. *IEEE T-PAMI*, 26:65–81, 2004. 1

[14] N. Komodakis, N. Paragios, and G. Tziritas. Mrf optimization via dual decomposition: Message-passing revisited. In *In ICCV*, 2007. 1

[15] N. Komodakis, G. Tziritas, and N. Paragios. Fast, Approximately Optimal Solutions for Single and Dynamic MRFs. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2007. 1

[16] X. Lan, S. Roth, D. Huttenlocher, and M. J. Black. Efficient belief propagation with learned higher-order markov random fields. In *ECCV*, 2006. 1, 7

[17] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *In Proceedings of Uncertainty in AI*, 1999. 1, 2

[18] Nvidia. Nvidia cuda programming guide (v 4.0), 2011. 6

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988. 1, 2

[20] G. Rong and T.-S. Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Symposium on Interactive 3D graphics and Games*, pages 109–116, 2006. 1, 4

[21] S. Roth and M. Black. Fields of experts: a framework for learning image priors. In *CVPR*, june 2005. 6, 7

[22] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23:309–314, August 2004. 1

[23] M. F. Tappen and W. T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. *ICCV*, 2:900, 2003. 1, 2

[24] Y. W. Teh, S. Osindero, and G. E. Hinton. Energy-based models for sparse overcomplete representations. *JMLR*, 4:1235–1260, October 2004. 6

[25] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *Information Theory, IEEE Transactions on*, 49(5):1120–1146, May 2003. 2

[26] Y. Xu, H. Chen, R. Klette, J. Liu, and T. Vaudrey. Belief propagation implementation using cuda on an nvidia gtx 280. In *Advances in Artificial Intelligence*, 2009. 1